BINDING OF PROCESSES IN NETWORK SYSTEMS

BACKGROUND OF THE INVENTION

FIELD OF THE INVENTION

The present invention relates to binding processes in a network system. More specifically, the present invention relates to ensuring that processes are bound to an active remote method invocation (RMI) process by monitoring the status of the RMI process.

RELATED ART

Administration of large, multi-server, computing environments is a field of growing interest as the number and size of large, multi-server computing environments grows. The field of multi-server system administration and management focuses on maintaining the physical operation of a multitude of computer systems, often referred to as nodes, connected in a network. This task includes a number of functions, including adding, modifying and removing nodes, users, tools, and roles; defining groups of nodes; authorizing users to perform operations on nodes; installing, maintaining and configuring hardware; installing and upgrading operating system and application software; and applying software patches, among other functions.

A typical network includes a plurality of nodes, which are managed by a service control manager (SCM) running on a central management server (CMS). The nodes comprise a service control manager cluster, and can be further organized into node groups. In a CMS, a plurality of processes, referred to as "daemons," are employed to perform tasks essential to run the network. The daemons are processes that perform tasks such as logging management actions by the SCM, managing users, and monitoring tasks assigned to nodes.

The daemons performing the above tasks may be located on differing JAVA® virtual machines (JVM), and remote method invocation (RMI) daemons are run in the network to allow daemons to communicate with one another. The RMI daemons serve as locators for daemons in the network, with agent daemons on each node accessing the RMI daemons in order to determine the network address, or universal resource locator (URL), for daemons in the network. A daemon in the network becomes accessible to users or other daemons by registering its URL in a URL list of an RMI daemon. This is commonly referred to as the daemon "binding" with the RMI daemon.

In conventional networks, if an RMI daemon becomes inactive for some reason, functioning daemons (and other processes) in the network remain bound to the inactive RMI daemon. In this case, it is not possible to communicate with the daemons bound to the inactive RMI daemon, because active RMI daemons would not include these daemons in their URL lists. In response to this situation, the network system restarts the daemons bound with the inactive RMI daemon. When the daemons restart, they are required to go through the process of registering with a new, active RMI daemon, which is time-consuming and introduces delay into the operation of the network.

Therefore, a need exists for a method of binding processes in a network that does not require restarting all of the processes bound with an RMI process when the RMI process becomes inactive.

SUMMARY OF THE INVENTION

The present invention overcomes the shortcomings of conventional methods and devices and may achieve further advantages not contemplated by conventional methods and devices.

According to a first aspect of the invention, processes in a network are each associated with a corresponding object, each object being capable of initiating a thread for monitoring the status of RMI. Processes having such an associated object are referred to as "parent processes." According to an embodiment of the invention, a method of binding the parent processes comprises binding a parent process with an RMI process, and calling an object associated with the parent process, the object initiating a thread. The thread performs the steps of monitoring the status of RMI processes, and rebinding the parent process with an active RMI process when the object determines that its parent process is not bound with an active RMI process.

According to the first aspect of the invention, parent processes in a network system need not be restarted when an RMI process becomes inactive, and may instead be automatically rebound with an active RMI process by the thread. Automatic rebinding of the parent process avoids delay and inconvenience to users of the network.

Other aspects and advantages of embodiments of the invention will be discussed with reference to the figures and to the detailed description of the preferred embodiments.

BRIEF DESCRIPTION OF THE FIGURES

Fig. 1 illustrates a network system according to an embodiment of the present invention.

Fig. 2 illustrates a portion of a network according to an embodiment of the present invention.

Fig. 3 is a flow chart illustrating the startup of processes and a watchdog thread associated with parent processes.

Fig. 4 illustrates the operation of a watchdog thread associated with a particular parent process.

Fig. 5 is a sequence diagram illustrating the operation of a watchdog thread for a parent process.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

A network system and a method for binding processes in a network system according to the present invention will be described below by way of preferred embodiments and with reference to the accompanying drawings.

Fig. 1 illustrates an exemplary network system 10 according to an embodiment of the present invention. The network system 10 comprises an SCM 12 running on a CMS 14, and a plurality of remote nodes 16 managed by the SCM 12 on the CMS 14. Together, the plurality of remote nodes 16 managed by the SCM 12 make up an SCM cluster 17. A group of remote nodes 16 may be further organized into node groups 18.

The CMS 14 may be, for example, an HP-UX 11.x server running the SCM 12 software. The CMS 14 includes a memory (not shown), a secondary storage device 141, a processor 142, an I/UX server 32, an input device (not shown), a display device (not shown),

and an output device (not shown). The memory, a computer readable medium, may include RAM or similar types of memory, and it may store one or more applications for execution by the processor 142, including the SCM 12 software. The secondary storage device 141 includes a data repository 26 for the SCM cluster 17, and a depot 30. The secondary storage device 141 may comprise a hard disk drive, a floppy disk drive, a CD-ROM drive, and other types of non-volatile data storage media. The CMS 14 also includes a web server 28 that allows web access to the SCM 12.

The processor 142 executes the SCM 12 software and other applications, which are stored in memory or in the secondary storage device 141, or received from the Internet or, in general, from another network 24. The SCM 12 may be programmed in Java®, and may operate in a Java® environment. Java® is an object-oriented program, and objects operating in a Java® Virtual Machine ("JVM") provide the functionality of the SCM 12. Object-oriented programming is a method of programming that pairs programming tasks and data into re-usable chunks known as objects - each object comprising attributes (*i.e.*, data) that define and describe the object. Java classes are meta-definitions that define the structure of a Java object. Java classes, when instantiated, create instances of the Java classes and are then considered Java objects. A detailed description of SCM is provided in, for example, ServiceControl Manager Technical Reference, HP part number: B8339-90019, which is hereby incorporated by reference, and which is available from Hewlett-Packard Company.

Generally, the SCM 12 supports managing an SCM cluster 17 from the CMS 14. Tasks performed on the SCM cluster 17 are initiated on the CMS 14 either directly or remotely, for example, by reaching the CMS 14 via a web connection 20. Therefore, a workstation 22 at which a user sits needs only the web connection 20 over the network 24 to the CMS 14, in order to perform tasks on the SCM cluster 17.

Fig. 2 illustrates a portion of the network system 10 according to an embodiment of the present invention. Fig. 2 illustrates the CMS 14, and one of the remote nodes 16 of the network system 10.

In the exemplary embodiment illustrated by Fig. 2, the functions of the SCM 12 are divided into a plurality of separate, long running, independently executing processes, which are referred to in the terminology of the UNIX systems community as "daemons." Fig. 2 shows four such processes running on the CMS 14: a Distributed Task Facility (for example, an ADTF@ process) process 210; a Log Manager process 212; a Domain Manager process 215; and an RMI process 205. For convenience, a process run by the CMS 14 can be generally referred to as a "management daemon," if the process is a daemon, or, more generally, as a "management process."

The Log Manager process 212 performs all of the functions of the SCM 12 necessary to maintain a log of the system management actions taken by the SCM 12. The log serves as an audit trail permitting an accounting of each step of each task performed by the SCM 12 on any of the nodes 16, node groups 18, or the SCM cluster 17, as well as on the CMS 14 itself. The Domain Manager process 215 performs the functions of the SCM 12 relating to the management of users and user groups on the SCM cluster 17. The Distributed Task Facility process 210 handles the assignment and monitoring of tasks assigned to be performed on each of the remote nodes 16. The RMI process 205 may be a JAVA® RMI process. Any of the processes 205, 210, 212, 215, 230 may be daemons.

Additional or different combinations of processes may be included in the CMS 14, and the configuration illustrated by Fig. 2 is intended to be exemplary.

The remote node 16 is illustrated as running a JAVA® RMI process 250, and an SCM Agent process (for example, an ASCM Agent@ process) 230. The remote node 16 is

illustrated as running the SCM agent process 230 and the RMI process 250. The CMS 14 also includes an SCM Repository 220. The RMI process 250 allows the processes 210, 212, 215, 230, which may be started in their own JVMs, to communicate with each other,even though they are in different JVMs.

In SCM environments such as those illustrated in Fig. 2, the RMI process 250 acts as an index, or locator. When one or more processes, such as the processes 210, 212, 215, 230, are started, the RMI process 250 stores the URL and object interface of each process that requires RMI functionality. The RMI process can respond to any process that may be looking for another one of the registered processes on the remote managed node 16 where the process can be found. For example, when the DTF process 210 needs to communicate with the agent process 230 to instruct it to perform an operation, it connects with the RMI process 250 and asks the RMI process 250 where the SCM agent process 230, of the given URL is located. The RMI process 250 responds with the interface object of the SCM agent process 230. The DTF process 210 may then connect with the SCM agent process 230 and communicate with it directly.

Before a process can be accessed in a network system, it must be registered with an active RMI process. A process registers with an RMI process by calling an RMI process initiated by an RMI object, which can be, for example, a JAVA® naming ("Naming") object, and providing its URL and interface object to the RMI process. In this manner, the process becomes "bound" with the RMI process, and other processes, users, or other entities, may then access the process through the RMI process. Each unique machine that has a process operating in a JVM requires an RMI process to be present.

Difficulties arise in conventional networks when an RMI process servicing a node becomes inactive for some reason, because processes bound with the inactive RMI process

would not be locatable. Instead, an attempt to access a process bound with an inactive RMI process would result in contact with an active RMI process that does not include the requested process in its bound URL list. Conventional networks resolve this problem by an inefficient restart (automatically performed by the operating system) of the processes bound with the inactive RMI process, so that the registered processes can again bind (or, "rebind") with the active RMI process.

The present invention overcomes the above shortcomings of conventional networks and achieves further advantages. According to an embodiment of the present invention, a process may be associated with its own object, which may be referred to as a "watchdog object," the watchdog object serving to initiate a thread, which may be conveniently referred to as a "watchdog thread." The watchdog thread monitors the status of RMI processes in order to determine whether the watchdog object's associated process is currently bound with an active RMI process in the network system 10. The watchdog thread acts to rebind its associated process with an active RMI process when the RMI process to which it is bound is no longer active. This function obviates the need to restart all of the processes bound with an RMI process when the RMI process becomes inactive. The network system 10 therefore operates more efficiently because processes become accessible as soon as an active RMI process becomes available to register the processes.

Processes in the network system 10 including an associated watchdog object may be conveniently referred to as "parent processes." Similarly, a daemon process including an associated watchdog object may be referred to as a "parent daemon." The watchdog thread may be employed in any process in the network system 10 that relies on RMI to communicate with other processes or users in the network system 10. Processes that may employ the watchdog thread include, for example, the processes 210, 212, 215, 230.

In Fig. 2, the functions of the SCM 12 are divided into separate processes to improve the reliability of the network. The configuration in Fig. 2, however, is merely illustrative, and other SCM network configurations employing RMI processes are also suitable for use with the present invention.

The operation of the watchdog thread will now be discussed with reference to Fig. 3. Fig. 3 is a flow chart illustrating the startup of the parent processes in the network system 10, and the startup of watchdog threads associated with parent processes in the network system 10.

In step S10, an RMI process is started. The RMI process can be started during installation of the SCM 12, or when other processes in the network are started. The other SCM processes in the network system 10 are then started in step S12.

In step S14, a watchdog object is called for each parent process, which initiates a watchdog thread for each parent process. In general, each parent process performs a method call to a watchdog object, which initiates a watchdog thread for that parent process. The watchdog thread monitors the status of the RMI process in order to determine whether the RMI process has registered its parent process. The operation of the watchdog thread, including the initialization call, will be discussed in further detail with reference to Figs. 4 and 5.

Step S16 illustrates the termination of a parent process. As discussed with reference to Fig. 4, if the terminated process is a parent process, the watchdog thread for the parent process may then be terminated, as its function is no longer required. One or more parent processes may be terminated to, for example, perform maintenance on the SCM 12.

Fig. 4 illustrates the operation of a watchdog thread associated with a particular parent process. In step S18, the watchdog thread obtains a bound URL list from the RMI process

via a list call. In step S20, the watchdog thread then determines whether its parent process's name is in the bound URL list of the RMI process. If the watchdog thread's parent process URL is in the bound URL list (i.e., the parent process is bound, or registered, with the RMI process) the watchdog thread returns to step S18, and periodically monitors the status of the RMI process for the presence of the parent process URL in the bound URL list.

If the watchdog thread's parent process URL is not in the bound URL list (i.e. the parent process is not bound, or registered, with the RMI process) the watchdog thread requests the RMI process to bind (via a rebind call) the parent process URL with the current, active RMI process (step S22). The parent process URL may be absent from the bound RMI list of an active RMI process if, for example, the RMI process to which the parent process was bound became inactive for some reason.

Because the parent process is now bound with the active RMI process, users, daemons, and other processes attempting to access the parent process can now communicate with the parent process. If the parent process were not rebound with an active RMI process, the active RMI process would report that the parent process was not bound to it, and the parent process would not be accessible.

In step S24, it is determined whether thread termination has been requested. The watchdog thread may be terminated, for example, when its parent process has been terminated.

Fig. 5 is a sequence diagram illustrating the operation of a watchdog thread for a parent process. In the exemplary embodiment illustrated by Fig. 5, the parent process is initiated by an object of class daemonImpl. In addition to an object, the term "daemonImpl" can represent the implementation of a daemon or other process.

The sequence diagram begins at the object named dtf of class daemonImpl, illustrated as dtf:DaemonImpl 300 in Fig. 5. The daemonImpl object dtf 300 initiates a parent process, having an associated object 304, named dtf, of class watchdog. The daemonImpl object dtf 300 first performs a synchronous rebind call to the RMI process initiated by the RMI object 302, which may include, for example, a JAVA® naming object. In the rebind call, the daemonImpl object dtf 300 provides its URL and interface object to the RMI process, thereby binding the daemonImpl object dtf 300 information with an active RMI process.

Once the daemonImpl object dtf 300 is bound with an active RMI process, the daemonImpl object dtf 300 performs an asynchronous initialize (init) call its associated watchdog object, dtf:Watchdog 304. Calling the watchdog object 304 starts a watchdog thread for the daemonImpl object dtf 300. The watchdog thread is illustrated as extending from the bottom of the watchdog object 304.

The watchdog thread includes a loop 308, in which a synchronous list call is performed in order to determine whether the URL of the parent process is in the bound URL list of an active RMI process. If the parent process URL is not listed with an active RMI process, the watchdog thread performs a rebind call to the RMI process in order to rebind the parent process with the active RMI process. The watchdog thread continues to perform list calls as long as the watchdog thread has not been terminated.

According to the above embodiment of the invention, if an RMI process becomes inactive for some reason, each parent process running a watchdog thread can quickly rebind with an active RMI process. Therefore, it is not necessary to restart every process upon inactivation of the RMI process.

The above sequence was described with reference to a parent process initiated by the daemonImpl object dtf 300, however the principles of the present invention apply to any daemon or other process having an associated object for generating a watchdog thread.

The steps of the above embodiments can be implemented with hardware or by execution of programs, modules or scripts. The programs, modules or scripts can be stored or embodied on one or more computer readable mediums in a variety of formats, such as source code, object code or executable code, for example. The code can be implemented in the Java® programming language, as described above, or in other programming languages. The computer readable mediums may include, for example, both storage devices and signals. Exemplary computer readable storage devices include conventional computer system RAM (random access memory), ROM (read only memory), EPROM (erasable, programmable ROM), EEPROM (electrically erasable, programmable ROM), and magnetic or optical disks or tapes. Exemplary computer readable signals, whether modulated using a carrier or not, are signals that a computer system hosting or running the described methods can be configured to access, including signals downloaded through the Internet or other networks.

The terms and descriptions used herein are set forth by way of illustration only and are not meant as limitations. Those skilled in the art will recognize that many variations are possible within the spirit and scope of the invention as defined in the following claims, and their equivalents, in which all terms are to be understood in their broadest possible sense unless otherwise indicated.